数理解析研究所講究録
906巻 1995年 154-161

MSCからのプロセス合成

臼井 伸幸 木村 成伴 富樫 敦 白鳥 則郎[†]

東北大学電気通信研究所 / 情報科学研究科[‡]

e-mail : {usui,kimura,togashi,norio}@shiratori.tohoku.ac.jp

概要

本論文では，Hennessy-Milner Logicからのプロセス合成アルゴリズムを利用し，複数のMSCを入力し，その振る舞いをするプロセスの合成法を提案し，合成されたプロセスとMSCとの関係とプロセスの等価性を議論する．

1 はじめに

近年，プロセスを形式的かつ代数的に扱うことが可能なプロセス計算に関する研究[7][3]が盛んに行われている．そこで筆者らは，μ-calculus [1]による論理式をプロセスの性質とみなし，与えられた性質からそれらを全て満たすプロセスを合成するアルゴリズム[4, 5]を提案した．

このアルゴリズムは，再帰を含むプロセスを有限ステップで合成することができるものである．

本稿では，このアルゴリズムを利用したプロセス合成法を提案する．具体的には，プロセスの性質としてシーケンス図を入力し，それをHennessy-Milner Logic[2]に変換するアルゴリズムを示す．このアルゴリズムは同期通信を行う複数のプロセスの性質を同時に出力するものである．

2 諸定義

2.1 プロセスの代数的形式化

以下ではアクション名全体の有限集合$\mathcal{L}$を仮定する．アクション(action)はシステムに実行される原始的動作の単位であり，外部から観測可能であると同時に制御可能であるとする．

定義 2.1 各アクション名$\alpha \in \mathcal{L}$に対して，同期通信の相手となる相補アクション名$\overline{\mathcal{L}}$とアクションの有限集合$\mathcal{A}$を以下のように定義する．

$$\overline{\mathcal{L}} = \{\overline{\alpha} | \alpha \in \mathcal{L}\} \quad \mathcal{A} = \mathcal{L} \cup \overline{\mathcal{L}}$$

本論文では，次のBNF表現で与えられるプロセスを考える．

$$p ::= \mathbf{0} \mid a.p \mid p + p$$

定義 2.2 ラベル付き遷移システムは，3項組$< S, \mathcal{A}, \rightarrow >$である．ここで$S$は状態の集合，$\mathcal{A}$はアクションの集合，$\rightarrow$は遷移関係であり，$\rightarrow \subset S \times \mathcal{A} \times S$として定義される． □

遷移関係について，$(s, a, s') \in \rightarrow$であるとき単に$s \xrightarrow{a} s'$と記すことにすると，遷移関係は$\rightarrow = \{\xrightarrow{a} | a \in \mathcal{A}\}$と表わすことができる．$s \xrightarrow{a} s'$は，状態$s$でアクション$a$を実行することができ，$a$の実行の結果，状態$s'$に遷移することを表わしている．

[†]Nobuyuki USUI, Shigetomo KIMURA, Atsushi TOGASHI, Norio SHIRATORI

[‡]{Research Institute of Electrical Communication ,Graduate School of Information Sciences} Tohoku University

定義 2.3 プロセスの動作的意味は，次の**遷移規則** *(transition rule)* によって与えられる．

$$\frac{}{a.p \xrightarrow{a} p} \qquad \frac{p \xrightarrow{a} p'}{p+q \xrightarrow{a} p'} \qquad \frac{q \xrightarrow{a} q'}{p+q \xrightarrow{a} q'}$$

□

このプロセス計算で記述したプロセスの例として，$p = a.\mathbf{0} + a.b.\mathbf{0}$ の遷移図を以下に示す．

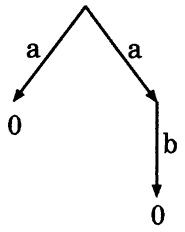

図 1: $p = a.\mathbf{0} + a.b.\mathbf{0}$ の遷移図

2.2 Hennessy-Milner Logic

プロセスの等価性を定める方法の中に，性質あるいは論理式を用いる方法がある．これは，二つのプロセスが等価であるとは，それぞれのプロセスが満足する性質 (論理式) が全く同じときである．そのような論理式として，Hennessy-Milner Logic[2] があり，以下のように定義する．

定義 2.4 論理式は次のように帰納的に定義される．
(1) T は論理式である．
(2) f, f' が論理式ならば，$f \vee f', \neg f$ は論理式である．
(3) f が論理式ならば，$\langle a \rangle f$ は論理式である．ここで $a \in \mathcal{A}$ とする． □

以下では，Hennessy-Milner Logic による論理式をプロセスの性質と考える．プロセス p が論理式 f を満たすことを $p \models f$ と書く．

定義 2.5 プロセスに対する論理式の充足性を次のように定義する．
(1) 任意のプロセス p に対して，$p \models T$ である．
(2) $p \models A_1 \vee A_2$ とは，$p \models A_1$ であるか，$p \models A_2$ であることを示す．
(3) $p \models \neg f$ とは，$p \not\models f$ である．ここで，$p \not\models f$ はプロセス p が f を満たさないことを示す．
(4) $p \models \langle a \rangle f$ とは，ある q が存在して，$p \xrightarrow{a} q$ かつ $q \models f$ である． □

定義 2.6 便宜的に以下の論理記号を導入する．
(1) $F \stackrel{\text{def}}{=} \neg T$.
(2) $A_1 \wedge A_2 \stackrel{\text{def}}{=} \neg(\neg A_1 \vee \neg A_2)$.
(3) $[a]A \stackrel{\text{def}}{=} \neg\langle a \rangle \neg A$. □

補題 2.7 定義 *2.6*で定めた任意の論理式は，論理演算子 $T, F, \wedge, \vee, \langle a \rangle, [a]$ だけを用いた論理式に等価変換できる． □

これらの論理式を用いて記述したプロセスの性質の例を紹介する．
(1) $p \models \langle a \rangle T$:$p$ において a が動作可能
(2) $p \models [a]F$:p は a を実行することは不可能である．
(3) $p \models [a]\langle b \rangle T$:$p$ は a が実行できるならば，必ずその後に b が動作可能である． □

2.3 メッセージシーケンス

メッセージシーケンス(MS)とは，複数のノード間におけるメッセージのやりとりの順序関係を記述するものである．

定義 2.8 メッセージの有限集合を $\mathcal{M} = \{(\alpha, modal) | \alpha \in \mathcal{L}, modal \in \{possible, condition, forbidden\}\}$ と定義する． □

直感的には，$(a,possible)$ はメッセージ a を送ることが可能であることを意味し，$(a,condition)$ はメッセージ a が送られたらという仮定を意味し，$(a,forbidden)$ はメッセージ a の伝送を禁止するという意味である．

特に誤解を与える恐れのないときは，$(a,possible)$,$(b,condition)$,$(c,forbidden)$ をそれぞれ，単に a^p, b^c, c^f と省略する．

定義 2.9 *MS* とは *3* 項組 $\langle \mathcal{N}, \mathcal{M}, R \rangle$ である．ここで，$\mathcal{N}, \mathcal{M}, R$ は以下のように定義される．
(1) $\mathcal{N}$ はノードの集合である．
(2) $\mathcal{M}$ はメッセージの集合である．
(3) R はノード間のメッセージパッシングの順序関係を表し，$R \subseteq P \times P$ である．ここで，$P \subseteq \mathcal{N} \times \mathcal{M} \times \mathcal{N}$ である． □

以下では特に断わることなく，(N_i, m, N_j) を m_{ij} と省略する．

2.4 メッセージシーケンス図

メッセージシーケンス図(MSC)はノードとメッセージからなる．メッセージには2種類あり，実線は伝達可能なメッセージ，破線は仮定，実線に × が付いている線(以下，禁止線と呼ぶ)は禁止されたメッセージを表わす．禁止線によって禁止される期間は禁止線が現われてから，次の実線，破線が現われるまでである．

補題 2.10 任意の *MSC* は等価な *MS* に変換することができる． □

図2に MSC の例を示す．この図は以下のような MS を表わしている．

$$
\begin{aligned}
\mathcal{M} &= \{(a, possible), (a, forbidden), (b, condition), (c, possible), (d, possible)\} \\
\mathcal{N} &= \{N1, N2, N3\} \\
P &= \{(N1, a^p, N2), (N2, b^c, N3), (N2, b^c, N1), (N1, c^p, N2), (N1, a^f, N2), (N3, d^p, N2)\} \\
R &= \{a^p_{12} < b^c_{21} = b^c_{23} < c^p_{12} < a^f_{12} < d^p_{32}\}
\end{aligned}
$$

図2の意味は，
(1) ノード $N1$ はメッセージ a をノード $N2$ に送ることができる．
(2) その後，ノード $N2$ がメッセージ b を $N1, N3$ に送る(この順番は任意である)ならば，
(3) ノード $N1$ はメッセージ c をノード $N2$ に返し，$N3$ は d を返す．
(4) ノード $N1$ はメッセージ c を送信後，メッセージ a を $N2$ に送ってはいけない．
ということである．

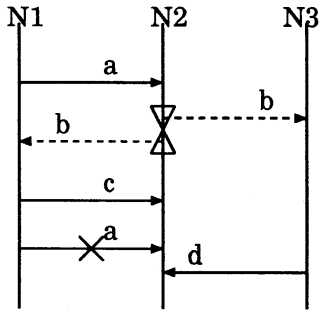

図 2: メッセージシーケンス図

2.5 試験システム

定義 2.11 ここでは，成功終了を表わす特別な試験者ωを仮定し，試験者は，次のように帰納的に定義する．

(1) $\mathbf{0}$,ωは試験者である．

(2) e が試験者であるとき，$a.e$ は試験者である．ここで，$a \in \mathcal{A} \cup \{1\}$ である．

(3) e_1, e_2 を試験者とすると，$e_1 + e_2$ は試験者である． □

定義 2.12 試験者とプロセスとの間の相互作用関係は，次の遷移規則によって与えられる．

$$\frac{e \xrightarrow{a} e' \quad p \xrightarrow{a} p'}{e\|p \rightarrow e'\|p'}(a \in \mathcal{A}) \qquad \frac{e \xrightarrow{1} e'}{e\|p \rightarrow e'\|p}(a \in \mathcal{A})$$

□

定義 2.13 試験システムは，4項組 $\langle P, E, \rightarrow, Suc \rangle$ で定義される．ここで，

(1) P は任意のプロセスの集合．

(2) E は任意の試験者の集合

(3) $\rightarrow$ は $E \times P$ 上の関係 $\rightarrow \subseteq (E \times P) \times (E \times P)$ であり，**相互作用関係** *(interacting relation)* と呼ばれる．

(4) $Suc \subseteq E$ は**成功集合** *(success set)* である．

ここでは，試験者の成功集合を $Suc = \{\omega\}$ とする． □

$Exp(e,p)$ で $e\|p$ から始まる試験全体の集合を表わすことにする．試験の結果は成功するか失敗するかのどちらかであり，$\top$ と $\bot$ でそれぞれ成功する試験結果と失敗する試験結果とする．したがって，$e\|p$ で始まる試験の結果 $Result(e,p)$ は，$\{\top, \bot\}$ の部分集合となる．

$$\top \in Result(e,p) \ : \ Exp(e,p)\text{が成功する試験を含むとき}$$

$$\bot \in Result(e,p) \ : \ Exp(e,p)\text{が失敗する試験を含むとき}$$

プロセスと試験との関係をこの $\top$ と $\bot$ を用いて，次のように書く．

$$p \ may \ e \ : \ \top \in Result(e,p) \quad p \ must \ e \ : \ \bot \notin Result(e,p)$$

定義 2.14 $MSC\gamma$ の各ノードを $N_i(\gamma)$ とするとき，$N_i(\gamma)$ における実線のメッセージ系列をそれに対応するアクション系列と成功終了，破線のメッセージを成功終了とみなし，$N_i(\gamma)$ に現われる禁止線を失敗終了とみなすことにより，$N_i(\gamma)$ と等価な試験者 $e(N_i(\gamma))$ を定義することができる． □

この may と $must$ の関係より，プロセスについて次のように新しい関係を定義する．

定義 2.15 $\langle P, E, \rightarrow, Suc\rangle$ が与えられたとき，

$$
\begin{aligned}
p \sqsubseteq_{may} q &: \text{任意の} e \in E \text{について，} p\ may\ e \text{ならば} q\ may\ e.\\
p \sqsubseteq_{must} q &: \text{任意の} e \in E \text{について，} p\ must\ e \text{ならば} q\ must\ e.\\
p \sqsubseteq q &: p \sqsubseteq_{may} q \text{かつ} p \sqsubseteq_{must} q.
\end{aligned}
$$

□

定義 2.16 プロセス p, q について，**may 試験等価** *(may testing equivalent)* $\simeq_{may}$ と **must 試験等価**
(must testing equivalent) $\simeq_{must}$ をつぎ のように定義する．

$$
\begin{aligned}
p \simeq_{may} q &: p \sqsubseteq_{may} q \text{ かつ } q \sqsubseteq_{may} p\\
p \simeq_{must} q &: p \sqsubseteq_{must} q \text{ かつ } q \sqsubseteq_{must} p
\end{aligned}
$$

□

$\mathcal{T}$ を試験者のある集合とし，$\mathcal{T}$ に関する前順序 $\sqsubseteq^{\mathcal{T}}_{may}$, $\sqsubseteq^{\mathcal{T}}_{must}$, $\sqsubseteq^{\mathcal{T}}$ を次のように定義する．ここで，p, q をプロセスとする．

定義 2.17 $\langle P, \mathcal{T}, \rightarrow, Suc\rangle$ が与えられたとき，

$$
\begin{aligned}
p \sqsubseteq^{\mathcal{T}}_{may} q &: \text{任意の} e \in \mathcal{T} \text{について，} p\ may\ e \text{ならば} q\ may\ e.\\
p \sqsubseteq^{\mathcal{T}}_{must} q &: \text{任意の} e \in \mathcal{T} \text{について，} p\ must\ e \text{ならば} q\ must\ e.\\
p \sqsubseteq^{\mathcal{T}} q &: p \sqsubseteq^{\mathcal{T}}_{may} q \text{かつ} p \sqsubseteq^{\mathcal{T}}_{must} q.
\end{aligned}
$$

□

また，$\sqsubseteq^{\mathcal{T}}_{may}$, $\sqsubseteq^{\mathcal{T}}_{must}$ に関しても $\sqsubseteq_{may}$, $\sqsubseteq_{must}$ と同様に $\simeq^{\mathcal{T}}_{may}$,$\simeq^{\mathcal{T}}_{must}$ を定める．

3 プロセスの合成

変換アルゴリズム 3.1は MS の集合が与えられと，それを定義 2.6によって定義される論理式の集合を出力する．さらに，出力された論理式をプロセス合成アルゴリズム [4] に与えることにより，与えられた MS の性質を満たすプロセスが合成される．

ユーザが与えるべきプロセスの性質は複数の MS である．

以下では，便宜的に MS，MS の集合，ノードを以下のように定め，それぞれ γ_i, Γ, $N_j(\gamma_i)$ と記述する．

$\gamma_i = \langle \mathcal{N}(\gamma_i), \mathcal{M}(\gamma_i), \rho(\gamma_i), \phi_{\gamma_i}\rangle$

$\Gamma = \{\gamma_i | 1 \leq i \leq n\}$

$N_j(\gamma_i) \in \mathcal{N}(\gamma_i)$ $\mathcal{N}(\gamma_i) = \{N_j(\gamma_i) | 1 \leq j \leq m\}$

$\mathcal{N}_j(\Gamma) = \{N_j(\gamma_i) | 1 \leq i \leq n\}$

また，アルゴリズムによって MS の集合 γ から得られたプロセスの集合を $\{p_j(\Gamma) | 1 \leq j \leq m\}$ と記述する．

3.1 変換アルゴリズム

以下に，変換アルゴリズムを示す．

アルゴリズム 3.1 変換アルゴリズム
入力: MS の集合 Γ
出力: A_i $(1 \leq i \leq m)$
(**) 内はコメント

```
begin
  (* メッセージシーケンスを読み込む *)
  ⟨N, M, R⟩ ← read-ms;
  k ← 1;
  while k ≤ m
    (* ノードの個数だけループする． *)
    begin
      A_i ← makeformula(N_i, ⟨M, R⟩);
      k ← k + 1;
    end
end

(*N_i に関する性質 A_i を合成する． *)
procedure makeformula(N_i, ⟨M, R⟩);
  begin
    (* メッセージがなければ，T を返す． *)
    if R = φ then return T;
    else
      begin
        (*MM に次のメッセージ集合を代入 *)
        MM ← get-next-message(R);
        R ← except-message(R, MM);
      (* さらに深いレベルの論理式を A に代入 *)
        A ← makeformula(N_i, ⟨M, R⟩);
        (*MM の全ての組み合せを作る *)
        A ← add-formula(N_i, M.M, MM, A);
        return A;
      end
  end

(*N_i に関する性質 A_i を合成する． *)
procedure add-formula(N_i, ⟨MM, CM, A⟩);
  begin
  if CM = φ then return A;
  else
    begin
      DM ← MM;
      m_jk ← get-member(CM);
      (*m_jk = (N_j, (act, modal), N_k)*);
      CM ← CM − {m_jk};
      A_s ← T;
      while MM ≠ φ
        begin
          DM ← DM − {m_jk)};
          A ← add-formula(N_i,
                ⟨MM − {m_jk}, CM, A⟩);
          (* メッセージが入力か出力かの判断 *)
          if modal = possible then
            g ← ⟨make-act(N_i, N_j, act, N_k)⟩A;
          else if modal = condition then
            g ← [make-act(N_i, N_j, act, N_k)]A;
          else if modal = forbidden then
          g ← [make-act(N_i, N_j, act, N_k)]F ∧ A;
          A_s ← A_s ∧ g;
          m_jk ← get-member(DM);
        end
      return A_s
    end
  end

(* メッセージに対するアクションの生成 *)
procedure make-act(N_i, N_j, act, N_k);
  (* メッセージが入力か出力かの判断 *)
  if N_i = N_j then return act_ik-bar;
  else if N_i = N_k then return act_ji
end.
```

□

定義 3.2 図 *3*のように同じアクション，同じ出力先のメッセージが禁止線，実線の順で隣り合って存在するメッセージシーケンス図を矛盾したメッセージシーケンス図と言う． □

定理 3.3 無矛盾なメッセージシーケンスはアルゴリズム *3.1*によって無矛盾な論理式に変換できる．
*(*証明*)* アルゴリズム *3.1*より自明． □

3.2 合成例

図 4をこのアルゴリズムに与えると，以下の論理式が出力される．

$$N1 \quad : \quad \langle \overline{a_{12}} \rangle [b_{21}] \langle \overline{c_{12}} \rangle [\overline{a_{12}}] F$$

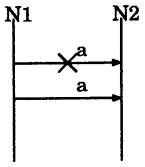

図 3: 実線と禁止線

$$
\begin{aligned}
N2 \ &: \ \langle a_{12}\rangle([\overline{b_{21}}][\overline{b_{23}}]\langle c_{12}\rangle([a_{12}]F \wedge \langle d_{32}\rangle T) \\
&\quad \wedge([\overline{b_{23}}][\overline{b_{21}}]\langle c_{12}\rangle([a_{12}]F \wedge \langle d_{32}\rangle T) \\
N3 \ &: \ [b_{23}]\langle\overline{d_{32}}\rangle T
\end{aligned}
$$

さらに，この論理式からプロセス合成アルゴリズムによって図5のような2つのプロセスが得られる．

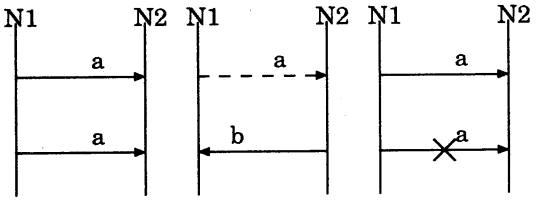

図 4: 入力する MSC

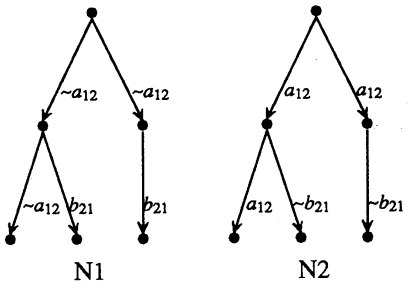

図 5: 合成されたプロセス

3.3 プロセスの正当性

求めているプロセスは入力したMSの集合Γにおけるそれぞれのノードの集合$\mathcal{N}_j(\Gamma)$の全ての振る舞いが可能であることが望まれる．そこで，ノードの動作的意味を考え，ノード動作式をプロセスと同様に定義して，ノード動作式を試験者としてプロセスの試験を行う．

定義 3.4 $MS\gamma$ におけるノード動作式 $\phi_\gamma(N(\gamma))$ に対応する試験者 $e(\phi_\gamma(N(\gamma)))$ を以下のように再帰的に定義する．

$$e(\mathbf{0}) = \omega \ , \ e(a.q) = a.e(q) \ , \ e(q_1 + q_2) = e(q_1) + e(q_2)$$

□

以下では，特に断ることなく $e(\phi_\gamma(N(\gamma)))$ を $e(N(\gamma))$, または $e(N)$ と記述する．例) 図2の3つのノードに対応する試験者は以下の通りである．

$$e(N1) = a_{12}.\omega \quad e(N2) = a_{21}.\omega \quad e(N3) = \omega$$

定理 3.5 アルゴリズムによって MS の集合 $\Gamma = \{\gamma_i | 1 \leq i \leq n\}$ から得られたプロセスの集合 $\{p_j | 1 \leq j \leq m\}$ は次の結果を満たす．

$$p_j(\Gamma)\ may\ e(N_j(\gamma_i)) \quad p_j(\Gamma)\ must\ e(N_j(\gamma_i)) \quad \text{ただし，} 1 \leq i \leq n, 1 \leq j \leq m$$

(証明) ノード動作式の定義より自明 □

以下では，MS の集合 $\Gamma = \{\gamma_i | 1 \leq i \leq n\}$ から得られる試験者の集合 $\mathcal{T} = \{e(N_j(\gamma_i)) | 1 \leq j \leq m\}$ に対して，次の性質を満たすプロセスの集合 $\{q_j | 1 \leq j \leq m\}$ を仮定する．

$$q_j \ may \ e(N_j(\gamma_i)) \ \ q_j \ must \ e(N_j(\gamma_i)) \ \ \text{ただし，} 1 \leq i \leq n, 1 \leq j \leq m$$

補題 3.6 アルゴリズムによって Γ から得られたプロセスの集合 $\{p_j | 1 \leq j \leq m\}$ は次の結果を満たす．

$$p_j(\Gamma) \simeq^{\mathcal{T}}_{may} q_j \ \ p_j(\Gamma) \simeq^{\mathcal{T}}_{must} q_j \ \ \text{ただし，} 1 \leq j \leq m$$

(証明) 定理 *3.5*と仮定より全ての $p_j(\Gamma), q_j$ は $e \in \mathcal{T}$ に対して $may, must$ の関係が成り立つので題意を満たす． □

定理 3.7 アルゴリズムによって Γ から得られたプロセスの集合 $\{p_j | 1 \leq j \leq m\}$ は次の結果を満たす．

$$p_j(\Gamma) \sqsubseteq q_j \ \ \text{ただし，} 1 \leq j \leq m$$

(証明) 仮定より，$p_j(\Gamma) \ may \ e$, $p_j(\Gamma) \ must \ e$ を満たす試験者 e は，$q_j \ may \ e$, $q_j \ must \ e$ を満たす．また，$p_j(\Gamma) \not{may} \ e$, $p_j(\Gamma) \not{must} \ e$ かつ $q' \ may \ e$, $q' \ must \ e$ となる試験者 e を用意すると，$q_j = p_j(\Gamma) + q'$ は仮定をみたし，さらに $p_j(\Gamma) \sqsubseteq q_j$ となるので証明は終了した． □

4 まとめ

本論文では，メッセージシーケンス図から Hennessy-Milner Logic のサブセットによる論理式を出力するアルゴリズムを与えた．これによって，プロセスの性質として，メッセージシーケンス図を枚挙することにより，並列に同期通信を行うプロセスを合成することができる．

今後の課題としては，デッドロックなどの検証，μ-calculus の不動点演算子への変換などが挙げられる．

参考文献

[1] Graf S., Sifakis J.: "A Logic for the Description of Non-deterministic Programs and Their Properties", Inf. and Contr., **68**, pp.254–270(1986)

[2] Hennesy M. and Milner R.: "Algebraic Laws for Nondeterminism and Concurrency", J. ACM., **32**, 1, pp. 137-161 (1985)

[3] Hoare C. A. R. : "Communicationg Sequential Process", Prentice Hall (1985).

[4] 木村成伴, 富樫敦, 白鳥則郎: "Synthesis Algorithm for Recursive Processes by μ-calculus ", 信学技報,**COMP93**,(1994-3).

[5] 木村成伴, 富樫敦, 野口正一 : "様相論理式による基本プロセスの合成アルゴリズム" , 信学論, **J75-D-I**, pp.1048–1061(1992).

[6] Kozen D.: "Results on the Propositional μ-calculus ", Theoret. Comput. Sci., **27**, pp.333–354(1983).

[7] Milner R.: "Communication and Concurrency", Prentice-Hall(1989).